23

Software Component Clustering and Retrieval: An Entropy-based Fuzzy *k*-Modes Methodology

Constantinos Stylianou and Andreas S. Andreou
Department of Computer Science, University of Cyprus
Cyprus

1. Introduction

Component-based software engineering is a stepwise software development process that relies on integrating small, independent software pieces into one larger fully-functioning system. However, a delay in any of the steps may negatively affect a project's schedule and, even worse, jeopardise its completion. Furthermore, given that the process is ultimately evaluation-oriented it is very difficult to find ways to accelerate certain stages in the process since developers are unwilling to compromise quality over time. Consequently, the only substantial means of reducing development time is to make components more easily and readily accessible to reusers in order to shorten the time taken for locating them. For this reason, component repositories have been introduced to store and organise software components. However, with an ever-increasing number of components, covering a wider variety of functionalities, there has been a parallel increase in the need for techniques to search and retrieve software components from repositories. Thus, the main motive of the research activity discussed in this chapter is to attempt to minimise the time necessary to discover any or all of the components to be reused in a component-based system by proposing an intelligent clustering methodology to aid the searching and retrieval of components stored in a repository.

To date there has been a number of attempts for clustering and classifying software components for reuse which have lead to the formation of various techniques. Several of these rely on structuring a formal specification of software components using classification methods based on natural language processing and documentation and retrieving components based on the resulting formal specification. Other attempts employ artificial intelligence methods for clustering software components, including genetic and evolutionary algorithms as well as self-organising feature maps. The methodology described in this chapter focuses on the utilisation of computational intelligence methods as the underlying clustering mechanism based on predefined lexical categories of software component attributes. Furthermore, the methodology incorporates a novel mechanism for efficiently searching and retrieving software components residing in a repository.

Specifically, the methodology employs an entropy-based fuzzy *k*-modes clustering algorithm, which has two main purposes. Firstly, to compute the number of clusters in a software component repository and to identify candidate initial cluster centres and secondly, to partition the software components into clusters with degrees of membership

and to locate the final cluster centres. Accordingly, the methodology continues to compare a user's search preference with the final cluster centres in order to isolate the cluster closest to the preference. The methodology concludes with an innovative retrieval technique to return those components from within the isolated cluster that is closest to the search preference by taking into account the degree of participation of each component in that cluster.

The evaluation of the efficiency and effectiveness of the methodology was carried out with two tests. The first test aimed at examining whether, with a given set of software components, the clustering mechanism adequately grouped similar software components. The second test aimed at assessing whether the retrieval mechanism appropriately fetched the most suitable software components based on a user's search preference. Observations of the results of the experiments proved that the combination of an entropy-based algorithm with a fuzzy *k*-modes algorithm as an intelligent partitioning scheme efficiently clusters software components into groups, which is significantly important as it deals with the difficult issue of separating data into an unknown number of sets. Furthermore, the results also showed that the approach exhibits a strong filtering mechanism since it will always return the most suitable components from a repository for the user to evaluate and select which of these will be later integrated into the software system. Through the experiments it was also noticed that the parameters used in the methodology may significantly affect the clustering and retrieval of software components.

The remainder of this chapter is organised as follows: Section 2 presents an overview of the component-based software engineering development process and also looks into various previous attempts carried out to deal with the problem of software component clustering, classification and retrieval. Subsequently, Section 3 describes the main concepts of clustering and gives a detailed explanation of two clustering algorithms employed within a three-step fuzzy logic-based method proposed by (Tsekouras et al., 2005). Next, Section 4 describes the proposed methodology, which uses the algorithms explained in Section 3, but as applied to the problem of clustering software components in a repository. This section also gives a description of the mechanism developed for the retrieval of the most suitable software components based on a user's search preference. Continuing, Section 5 provides an analysis of the methodology's assessment procedure as well as an evaluation of the results obtained. Finally, in Section 6 the main strengths and weaknesses of the methodology are discussed in addition to possible alternatives or modifications that could be carried out in future research efforts.

2. Component-based software engineering overview

This section explores the component-based software development process and reviews various attempts and techniques that have previously been developed relating to the issue of component clustering and classification, in addition to searching and retrieval.

2.1 Software reuse, components & component-based software engineering

The idea of reuse is not recent. Ever since programming began, developers have been reusing previously implemented code segments, functions and procedures to build new software systems. However, it was not until 1968 at the NATO Software Engineering Conference at Garmisch, that the notion of "software reuse" was first formalised in an attempt to tackle the software crisis. There, Douglas McIlroy (Naur & Randell, 1969) argued

that systems should be built from "reusable software components" and over the years there have been many definitions given for the term. One of which states that software components are "independent deployable software entities with a certain functionality which can be composed into larger systems by means of dynamically discoverable, immutable interfaces following standardised conventions" (Naedele, 2000). By this definition, it is possible for a software system to be comprised entirely of smaller, individual units, each of which performs a specific functionality in the system. Subsequently, the introduction of software components also gave rise to a new "integrate, not implement" style of software development named component-based software engineering. Systems built from reusable components following this approach are said to be component-based systems and over the years the software engineering industry has witnessed the expansion of the concept of "software reuse" to include not only code, but requirements, designs, architectures, test-cases and many more reusable assets of software engineering (Frakes, 2007).

2.2 Trends in component-based software engineering

Recent years have seen an increase in the standardisation and adoption of component reuse approaches to software development due to the shift of developers/programmers towards minimising the re-creation of code. For instance, object-oriented programming with its mechanisms for methods, inheritance and dynamic binding is one such approach that adopts reuse. Minimising the re-creation of code in turn has prompted software houses to prefer to employ component-based methodologies for developing systems over traditional ones, offering them a much simple and accessible solution. Furthermore, the component-based software development process has been improved with the introduction of component repositories for storing large volumes of software components.

With components becoming more readily available and having in mind that this will allow them to remain active in an ever-competitive industry, more companies are now willing to undergo the cost of buying software components. However, as explained in the next subsection, the process of development based on component reuse is a reasonably rigorous approach which emphasises the evaluation of components, their correct integration into the software system, and the components' maintenance.

2.3 The component-based software development process

As with any other disciplined software development process, there are certain steps that must be followed for successful software component reuse. Specifically, the component-based software development process primarily consists of four steps (Brown & Wallnau, 1996):

1. Component qualification (sometimes referred to as suitability testing)
2. Component adaptation
3. Assembling components into systems
4. System evolution

Component qualification is the first step and is considered as "a process of determining fitness for use of previously-developed components that are being applied in a new system context" (Haines et al., 2007). This step is the most determinative of the four, because the success of component reuse is largely based on whether the correct component is selected for reuse. It can be logically separated into two phases – discovery and evaluation –

whereby the former involves searching and retrieving the most suitable software components while the latter adopts criteria to examine the "non-technical" aspects such as quality, usability, developer's credibility and many more.

Once a component has been selected for reuse, the second step requires component adaptation to be carried out. Components may need to be configured before being integrated into the system given that they are generally built to be used in different contexts (Haines et al., 2007), which may lead to a greater chance of conflicts between components.

The third step of the process entails the components assembly into the system. This can be viewed as the integration of the components by binding the individual components with a well-defined infrastructure. The infrastructure forming the system can be based on several architectural styles, including, databases, blackboards, and the more popular, object request brokers (ORBs) (Haines et al., 2007).

The final step of the process is system evolution, similar to the concept of software maintenance in traditional development life-cycles. This step deals with the evolving and upgrading of a system by replacing erroneous and outdated components with newer and more advanced components. This unavoidably will require the latest components to be tested and validated before replacing the previous one. If a component needs replacing, but there is no updated version for it, then the process will reiterate to the first step to discover and evaluate components again (Haines et al., 2007). As a result, this step may end up being the longest with respect to the process as a whole.

The process explained above can only be considered productive if the time taken to successfully reuse a software component is less than the time it would take for a developer to implement the particular functionality from scratch – and this extends to the development of the system as a whole. This limitation coincides with one of the most challenging issues facing software houses, that is: being able to deliver projects within the predetermined deadline, and without exceeding the budget or compromising the system's quality.

This is the point to which attention is drawn as the methodology proposed aims to provide a solution for minimising the development time of component-based systems without having adverse affects on productivity, effectiveness, and, above all, quality. In order to minimise the development time, it is imperative to identify which parts of the process are the most time-consuming and in turn try to reduce the duration required to complete them. From the steps explained above, it can be observed that component discovery is one of the lengthiest stages in the process. This is because developers struggle to locate the most appropriate components to evaluate. If it is possible to find a way to reduce the time spent by developers searching and retrieving components, then component-based software development process will become more productive, efficient and reliable. Furthermore, more time can then be allocated to the other steps of the development process, for instance, to evaluate the retrieved components further on in the development process, whose duration cannot or, rather should not, be reduced. Also, more time will be available for the proper adaptation and assembling of the software components into the system's framework.

2.4 Previous attempts of component clustering, search & retrieval

Over the years, various attempts have been made to aid developers in their search and retrieval of software components for reuse.

Firstly, (Prieto-Díaz & Freeman, 1987; Prieto-Díaz, 1991) attempted to tackle the issue of software component classification with an informal method whereby experts extract facets from keywords contained in software components, and subsequently using these facets to describe technical and non-functional features of components. The method then continues to retrieve components by employing a weighted conceptual graph to match users' queries with software components taking into account the facets extracted by the experts. Another informal method includes natural language processing in queries for the retrieval of software components. Such techniques include semantic networks and free-text analysis for automatic keyword extraction (e.g., Girardi & Ibrahim, 1995; Sugumaran & Storey, 2003; Yao & Etzkorn, 2004). Formal methods, alternatively, aim to cluster and classify components by providing a specification for software components (Chu et al., 2000; Nakkrasae & Sophatsathit, 2002). Additionally, (Nakkrasae et al., 2004) expand the use of a formal specification for classification by combining a fuzzy subtractive clustering technique. A more recent attempt by (Chang et al., 2005) implements a scheme for the automatic clustering of use-cases, actors, classes and other elements of object-oriented modelling into candidate components.

There are also some noticeably distinct approaches that employ computational and artificial intelligence methods. For example, (Wang et al., 2004) use self-organising feature maps in order to cluster a software component catalogue, while (Andreou et al., 2006) make use of genetic and evolutionary algorithms to cluster software components. The proposed methodology deals with the issue of component clustering and retrieval from a similar perspective as the last two abovementioned studies. By employing computational intelligence and fuzzy logic techniques the methodology focuses on the construction, firstly of the principal clustering mechanism and secondly, of the mechanism required for searching and retrieving the most adequate components from repositories.

3. Clustering algorithms

Clustering is a form of unsupervised learning that aims to analyse and organise data into groups based on their similarity (Jain et al., 1999). For this reason, clustering is widely used in various problem-solving and decision-making applications, such as document retrieval, marketing research, genotype assignment, insurance fraud identification, image segmentation, and city-planning, to name a few. Over the years, there have been many methods and techniques developed to perform cluster analysis. The main types of clustering methods that exist include (Han & Kamber, 2000):

- Partitional clustering: This is one of the most common clustering methods, which aims to partition data into a finite number of clusters based on various distance measures and criteria. Examples include *k*-means, PAM and CLARANS.
- Hierarchical clustering: This method transforms a dataset into a hierarchical tree structure. Hierarchical clustering models can adopt a bottom-up (agglomerative) approach, such as in AGNES, which iteratively merges clusters into larger clusters. On the other hand, they can adopt a top-down (divisive) approach, for instance as in DIANA, by repeatedly splitting clusters into smaller clusters. In either approaches merging or splitting ceases when a termination condition is satisfied (Kauffman & Rousseeuw, 1990).
- Density-based clustering: This type of clustering takes into account the fact that data objects may form clusters with arbitrary shapes and requires a cluster to continue to

grow on condition that the number of objects assigned to it (i.e., its density) exceeds a minimum threshold value. Examples of density-based methods can be found in the DBSCAN (Ester et al., 1996) and OPTICS (Ankerst et al., 1999) models.

- Grid-based clustering: This approach creates a grid structure from a finite number of cells on the data object space and only needs to operate clustering on this structure rather than on all the data objects per se. For this reason it requires very little processing time. Some of the most popular grid-based clustering methods include CLIQUE (Agrawal et al., 1998), STING (Wang et al., 1997) and WaveCluster (Sheikholeslami et al., 2000).
- Model-based clustering: This form of clustering assumes that each cluster has its own model and attempts to determine the best fit of the data to that given model.

From the above list, it is apparent that clustering can take many forms and furthermore, it offers a wide variety of possible solutions to data mining issues. For example, it has the ability to discover trends in large datasets and can be used to identify potential outliers that are inconsistent with the remaining data.

In order to improve the process of searching and retrieving components it was decided that first and foremost it was imperative to partition the software components in a repository to reduce, or rather, eliminate less suitable components. The component clustering process, therefore, is the focal point of the suggested methodology, which adopts a strategy to cluster components of a repository based on a similar approach used by (Tsekouras et al., 2005). Here, however, the authors applied their technique to help in the analysis of cultural data, in particular, ethnocultural identity (Cushner & Brislin, 1997; Tsekouras et al., 2005) which is a major field in study of cross-cultural adaptation. The subsequent subsections take an in-depth look at the clustering algorithms adopted and present the advantages for their use in the suggested methodology as well as their underlying limitations.

3.1 Entropy-based clustering

Entropy-based clustering (Yao et al., 2000) is a technique that was first introduced in 1998, which seeks to arrange similar data objects into clusters based on their total entropy values. The goal of the technique is to traverse the dataset in just one pass to determine the number of clusters present and also to identify the location of the cluster centres. The basic idea is that if a data object has many surrounding objects, then its total entropy value will be relatively lower and can therefore be considered as a strong candidate for a cluster representative. The entropy value calculated for each data object is based on a predefined similarity measure. The data object achieving the lowest total entropy value is selected as the first cluster centre. At this point, the algorithm uses a parameter, β, representing a threshold of similarity or association (Yao et al., 2000). Any data objects with high similarity to the recently selected cluster centre (i.e., data objects with a similarity value higher than the threshold β) are removed from the dataset. The reasoning behind this is that since these data objects are similar they should stop being considered as potential clusters, and instead be assigned to the cluster they are highly similar to. Once these data objects are removed from the dataset, the number of clusters is increased and the data object with the next least total entropy value is selected and the procedure repeats until there are no objects left in the dataset.

3.1.1 Entropy-based clustering notations

Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n data objects described by m attributes. The entropy value, H_{ij}, of two data objects, X_i and X_j, is defined by:

$$H_{ij} = -E_{ij}\log_2(E_{ij}) - (1-E_{ij})\log_2(1-E_{ij}) \quad (1)$$

where $i \neq j$. E_{ij} is a similarity measure between X_i and X_j and is calculated using:

$$E_{ij} = e^{-\alpha D_{ij}} \quad (2)$$

where D_{ij} is the distance between data objects X_i and X_j, and a is calculated automatically by:

$$\alpha = -\ln(0.5)/\bar{D} \quad (3)$$

where $\bar{D}$ is the mean distance among the pairs of data objects in a hyperspace. From Equation (1), the total entropy value of data object X_i with respect to all other data objects is computed as:

$$H_i = -\sum_{\substack{j=1 \\ i \neq k}}^{n} \left[E_{ij}\log_2(E_{ij}) - (1-E_{ij})\log_2(1-E_{ij}) \right] \quad (4)$$

Finally, let $Z_l = \{Z_{l,1}, Z_{l,2}, \ldots, Z_{l,m}\}$ represent a cluster centre, which is assigned the data object that achieves the least entropy value after each iteration.

3.1.2 Entropy-based clustering algorithm

The entropy-based clustering algorithm is presented in Fig. 1.

Algorithm: **Entropy-based Clustering**

1. Select threshold of similarity, β, and set the initial number of clusters $c = 0$.
2. Determine the total entropy values H for each data object in X based on Equation (4).
3. Set $c = c + 1$.
4. Select the data object X_{min} with the least entropy H_{min} and set $Z_c = X_{min}$ as the c^{th} cluster centre.
5. Remove X_{min} and all data objects having similarity with X_{min} greater than β from X.
6. If X is empty then stop; otherwise go to step 3.

Fig. 1. Entropy-based clustering algorithm

The key reasons for using an entropy-based clustering algorithm are two-fold. It is effectively capable of computing the number of clusters in a given dataset, in addition to discovering which objects are candidates for cluster centres. Secondly, it is highly efficient due to the fact that the entropy values of data objects are required to be calculated only once and after a cluster centre is determined, data objects are iteratively removed from the dataset.

3.2 Hard and fuzzy *k*-modes algorithms

In 1998, (Huang, 1998) introduced the *k*-modes algorithm as an alternative method of clustering objects in a dataset. This partitional clustering method is specifically designed to handle categorical data, a limitation posed by the hard and fuzzy *k*-means algorithms. There

are 3 basic modifications to the original algorithm. Firstly, the distance measure between two objects is altered to a simple matching of the attributes of the dataset as opposed to the squared Euclidean distance. Secondly, the cluster centres are defined by the modal value of each attribute instead of the mean value. Finally, computation of the modes utilises a frequency-based method applied on every category of the dataset's attributes. Despite these alterations, the clustering method still keeps the efficiency of the *k*-means algorithm, especially for large datasets. As previous, the objective of the algorithm is to minimise its cost function in order to separate the dataset into clusters.

3.2.1 *k*-modes notations

Clustering by *k*-modes adopts extremely similar notations from those of *k*-means clustering. However, since the attributes in the latter consist of numeric data – and consequently their domains – modifications are required to accommodate the use of categorical values. Specifically, let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n data objects. Each of these objects can be described by a set of m attributes $A_1, A_2, \ldots, A_m$, and each attribute, A_j, can take any value from the attribute domain $\mathrm{DOM}(A_j) = \{a_j^{(1)}, a_j^{(2)}, \ldots, a_j^{(nj)}\}$ where n_j is the number of category values possible for attribute A_j, for $1 \le j \le m$. Data objects can therefore be logically represented by a conjunction of attribute-value pairs $[A_{i,1} = x_{i,1}] \wedge [A_{i,2} = x_{i,2}] \wedge \ldots \wedge [A_{i,m} = x_{i,m}]$, where $x_{ij} \in \mathrm{DOM}(A_j)$, for $1 \le j \le m$ (Kim et al., 2004) and hence this is extended so that each X_i can be represented by a vector $[x_{i,1}, x_{i,2}, \ldots, x_{i,m}]$ for $1 \le i \le n$. Finally, let a cluster centre be symbolised by $Z_l = \{z_{l,1}, z_{l,2}, \ldots, z_{l,m}\}$ for $1 \le l \le k$. To minimise a cost function $F(W, Z)$ (Bezdek, 1980) therefore requires that:

$$F(W, Z) = \sum_{l=1}^{k}\sum_{i=1}^{n} w_{li}^{\alpha} d(Z_l, X_i) \tag{5}$$

subject to:

$$0 \le w_{li} \le 1, \quad 1 \le l \le k, 1 \le i \le n \tag{6}$$

$$\sum_{l=1}^{k} w_{li} = 1, \quad 1 \le i \le n \tag{7}$$

$$0 \le \sum_{i=1}^{n} w_{li} \le n, \quad 1 \le l \le k \tag{8}$$

where k ($\le n$) is a predefined number of clusters, $W = [w_{li}]$ is a $k \times n$ partition matrix, $Z = \{Z_1, Z_2, \ldots, Z_k\}$ is the set of cluster centres, and $d(\cdot, \cdot)$ is some measure of distance between the two objects. The fuzziness exponent, $a \in [1, \infty)$ found in Equation (5) identifies whether hard ($a = 1$) or fuzzy ($a > 1$) *k*-modes clustering occurs.

3.2.2 The *k*-modes dissimilarity function

In contrast with *k*-means clustering, which calculates the dissimilarity of objects with the Euclidean norm, the dissimilarity function comprises of matching the category values of attributes comprising the objects. If two objects share the same category in an attribute, then

the distance for that attribute is assigned a value of zero, symbolising that there is no difference. Whereas if the objects do not have the same category value for an attribute, then the difference is allocated a value of one, indicating that a difference exists. By employing such generalised Hamming distance (Kohonen, 1987; Hunag, 1998), it is possible to see how close two objects are by summing the number of mismatches of category values. Hence, the smaller the number of mismatches, the nearer the objects are. The formal representation of the dissimilarity function is as follows:

Let $X_1 = [x_{11}, x_{12}, \ldots, x_{1m}]$ and $X_2 = [x_{21}, x_{22}, \ldots, x_{2m}]$ be two data objects of X defined by m attributes. The dissimilarity between the two objects, $d(X_1, X_2)$, is denoted by:

$$d(X_1, X_2) = \sum_{j=1}^{m} \delta(x_{1j}, x_{2j}) \tag{9}$$

where:

$$\delta(x_{1j}, x_{2j}) = \begin{cases} 0, & x_{1j} = x_{2j} \\ 1, & x_{1j} \neq x_{2j} \end{cases} \tag{10}$$

The dissimilarity function in Equation (9) is then used to (re)assign a data object to a cluster. Accordingly, in the case of the hard *k*-modes algorithm, if object X_i yields the shortest distance with centre Z_l in a given iteration, this is represented by setting the value at the nearest cluster to 1 and the values at the rest of the clusters to 0 in the partition matrix W. Formally, for $a = 1$:

$$\hat{w}_{li} = \begin{cases} 1, & \text{if } d(Z_l, X_i) \leq d(Z_h, X_i), \quad 1 \leq h \leq k \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

On the other hand, in the case of the fuzzy *k*-modes algorithm, for $a > 1$, the partition matrix W is given by:

$$\hat{w}_{li} = \begin{cases} 1, & \text{if } X_i = Z_l \\ 0, & \text{if } X_i = Z_h, \; h \neq l \\ \dfrac{1}{\displaystyle\sum_{h=1}^{k} \left[\dfrac{d(Z_l, X_i)}{d(Z_h, X_i)} \right]^{1/(\alpha-1)}}, & \text{if } X_i \neq Z_l \text{ and } X_i \neq Z_h, 1 \leq h \leq k \end{cases} \tag{12}$$

for $1 \leq l \leq k$, $1 \leq i \leq n$. This means that if a data object shares the same values for all attributes with a particular cluster centre, then it will be assigned wholly to that cluster and not at all to the rest. Conversely, if a data object is not completely identical to any of the cluster centres, then the data object is assigned a membership degree to each cluster, which also solves the constrained optimisation problem of Equation (7) (Tsekouras et al., 2005).

Equation (6), however, does not emphasise any importance of a category in an attribute. For example in a hospital database, the category value "YES" of the attribute "SMOKER" may

be considered more important than the category value "NO". For this reason, (He et al., 2007) provides various methods to assign weights not only to the attributes but also to the categories in attributes to properly reflect the reality of dissimilarity between data objects. Alternatively, in order to take into account the frequencies of the categories in the dataset, (Huang, 1998) proposed a chi-squared distance, which gives greater importance to less frequent categories of an attribute. Likewise, (San et al., 2004) suggest integrating the relative frequency of a category in the distance function. In either case, the partition matrix, $W = [w_{li}]$, is computed by using Equation (11) for hard *k*-modes clustering and Equation (12) for fuzzy *k*-modes clustering.

3.2.3 The *k*-modes update function

Updating the centres in *k*-modes clustering uses a frequency-based method that basically assigns the most frequent category of each attribute as representative of the cluster. It should be noted, however, that the newly computed mode of a cluster does not necessarily mean it is an element of that cluster. Strictly speaking, every cluster, $Z_l = [z_{l,1}, z_{l,2}, \ldots, z_{l,m}]$ for $1 \le j \le m$, is updated so that:

$$\hat{z}_{l,j} = a_j^{(r)} \in \mathrm{DOM}\left(A_j\right) \tag{13}$$

where, in the case of hard *k*-modes clustering, $a_j^{(r)}$ satisfies:

$$\left|\left\{w_{li} \middle| x_{i,j} = a_j^{(r)},\ w_{li} = 1\right\}\right| \ge \left|\left\{w_{li} \middle| x_{i,j} = a_j^{(t)},\ w_{li} = 1\right\}\right|, \quad 1 \le t \le n_j \tag{14}$$

for $1 \le l \le k$. It can be observed from Equation (13) that the mode is neither unique. It is possible for two categories to tie as the most frequent in an attribute. In such cases, the algorithm will arbitrarily assign the first category that achieves the maximum frequency (Huang & Ng, 1999) for the attribute in its cluster.

In fuzzy *k*-modes clustering, the general idea of the update function remains the same – the mode is decided based on the category achieving highest frequency. The difference lies in the computation of the frequency since there are no crisp values, (i.e., zeros or ones) in the partition matrix but membership values. For this reason the update function is represented as Equation (14) though $a_j^{(r)}$ must now satisfy:

$$\sum_{i, x_{i,j} = a_j^{(r)}} w_{li}^{\alpha} \ge \sum_{i, x_{i,j} = a_j^{(t)}} w_{li}^{\alpha}, \quad 1 \le t \le n_j \tag{15}$$

By Equation (14) the category assigned to attribute A_j of cluster Z_l is the category which "achieves the maximum of the summation of w_{li} to cluster Z_l over all categories" (Huang & Ng, 1999). Nevertheless, it is still possible for two or more categories to tie as most frequent in the attribute.

3.2.4 The *k*-modes algorithm

The *k*-modes algorithm performs exactly like the *k*-means algorithm in that it attempts to minimise the cost function, firstly by fixing Z to determine W and secondly, fixing W to determine Z. The algorithm required to perform this is outlined in Fig. 2.

Algorithm: ***k*-Modes Clustering**

1. Select k initial clusters randomly $Z^{(1)} = \{Z_1^{(1)}, Z_2^{(1)}, \ldots, Z_k^{(1)}\}$.
2. Determine $W^{(1)}$ by selecting either Equation (11) or Equation (12) for the type of k-modes clustering, such that $F(W, Z^{(1)})$ is minimised.
3. Set $t = 1$.
4. Determine $Z^{(t+1)}$ based on Equation (13) satisfying either Equation (14) for hard k-modes clustering or Equation (15) for fuzzy k-modes clustering, such that $F(W^{(t)}, Z^{(t+1)})$ is minimised.
5. If $F(W^{(t)}, Z^{(t+1)}) = F(W^{(t)}, Z^{(t)})$ then stop; otherwise go to step 6.
6. Determine $W^{(t+1)}$ using the same equation used in step 2, such that $F(W^{(t+1)}, Z^{(t+1)})$ is minimised.
7. If $F(W^{(t+1)}, Z^{(t+1)}) = F(W^{(t)}, Z^{(t+1)})$ then stop; otherwise set $t = t + 1$ and go to step 4.

Fig. 2. The k-modes clustering algorithm

Since the k-modes clustering algorithm is adapted to function on categorical data, it is a suitable candidate for the clustering algorithm required for the methodology proposed in the next section. Furthermore, because of the disorderly and chaotic configuration of components in a repository, the fuzzy k-modes clustering algorithm is preferred over hard k-modes clustering to allow for levels of ambiguity in a repository's configuration.

4. Proposed methodology

This section presents the proposed methodology for the clustering and retrieval of software components from a component repository, indicating any necessary parameters as well as computed outputs. The three-step approach is defined by: (i) the clustering procedure including any pre-processing activities required, (ii) the input of the user's search preference and the isolation of the subset of software components in which the actual search will be carried out, and (iii) the retrieval of the most suitable components from the repository.

4.1 Software component attributes

Software components in repositories are composed of many different behavioural, functional, or structural attributes (Nakkrasae & Sophatsathit, 2002). The main attributes chosen to comprise the software components in the methodology were selected based on an earlier attempt carried out by (Andreou et al., 2006), which features software components described by both technical and non-functional characteristics. Specifically, the software component attributes are: component domain and specific functionality, platform, implementation language, operating system independence, synchronisation, visibility of implementation, price range, processor, memory and disk utilisation, binding, data encryption, data open format compatibility and finally, password protection. Furthermore, each of the 15 selected attributes is assigned a predefined domain that consists of categorical values (or categories). It is important to note that any other schemes attempting to employ the suggested methodology are not bound by the use of only these 15 attributes. In fact, other attributes may be included or present ones removed depending on the quality of the information found in the component repository.

4.2 Roles and parameters of the methodology

There are two main roles in the methodology. Firstly, there is that of the "application administrator" whose is responsible for configuring the nonuser-related parameters required by the clustering mechanism. The second role is that of the "user", who is required to provide their search preference and the level of confidence (covered in Subsection 4.5) to control the level to which the results should satisfy their search. Furthermore, as the approach is loosely based on a client-server model, the application administrator is involved at the server-side and conversely, the users' interactions and searches take place at the client-side.

4.3 Clustering procedure

The purpose of this step, as mentioned previously, is to avoid having to locate candidate components from the whole of the repository, but instead to isolate a part of it and perform the search in that part. Therefore, the first step in the methodology entails the clustering of the software components into groups based on the 15 features listed in Subsection 4.1. However, a problem arises since the number of groups to be formed is initially unknown. Hence, the clustering technique to be employed must be capable of determining the number of clusters in the repository automatically, while still being able to respond to and cope with an extremely large volume of high-dimensional data. The partitioning technique used therefore, utilises a clustering approach suggested by (Tsekouras et al., 2005), which consists of pre-processing data using the entropy-based clustering algorithm before feeding the results into the connecting fuzzy *k*-modes clustering algorithm.

4.3.1 Entropy-based clustering

The pre-processing begins by executing the entropy-based clustering algorithm explained in Section 3.1 on the repository in order to compute the number of clusters present as well as the candidate cluster centres. The entropy value of each component is calculated by evaluating the dissimilarity of each component with regards to all others. Here, the categorical nature of component attribute-types leads to using the Hamming distance as the most suitable dissimilarity measure between two components. Subsequently, a new cluster is formed assigning the component achieving the lowest entropy value. Next, all components achieving similarity equal or above the value of the threshold, β (selected by the application administrator) are assigned to the newly created cluster and cease from being considered as potential centres. At the end of this stage, the entropy-based clustering algorithm will have computed the parameter k along with the k initial cluster centres for the fuzzy *k*-modes algorithm to apply.

4.3.2 Fuzzy *k*-modes clustering

With the completion of the pre-processing stage, the clustering procedure executes the fuzzy *k*-modes clustering algorithm using as inputs the value of k and the initial cluster centres. The final input required is the fuzziness exponent, a, and this is selected by the application administrator in order to set the level of fuzziness to be considered in the clustering of the software components. For each component, the measure of similarity between each cluster centre is computed by using a simple pattern match of the attributes of the components to the cluster centres to determine the distance. Each component will then be assigned to all clusters. However, the higher the similarity to a cluster centre, the higher the participation

(level of membership) the component will have in the respective cluster. After each iteration (i.e., after all components have been assigned), using the appropriate update functions of Equations (13)-(15), the cluster centres are revised resulting in new cluster centres. Specifically, the new centres are found by computing the mode from the categories of component attributes that achieve the highest summation of membership degrees in the cluster. The process of assignment and update repeats until the algorithm converges. The end results will be a partition matrix holding the membership degrees (easily convertible to percentages) of components to clusters as well as the finalised cluster centres. Both these elements will be used in the subsequent steps. Table 1 presents an example of a component partition matrix indicating the degree of membership of each component in all clusters. The values in boldface represent the highest membership degree of a component, relative to the cluster in which it achieved the degree. As shown in the matrix it is possible for a component to achieve highest membership in more than one cluster, as for example components C4 and C6.

	Component Assignments									
	C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**
Cluster Z1	0.13	0.50	**0.94**	**0.38**	0.17	**0.44**	**0.79**	0.04	0.02	0.13
Cluster Z2	0.08	**0.69**	0.03	0.18	0.17	0.09	0.10	**0.86**	0.03	0.08
Cluster Z3	0.22	0.20	0.01	0.06	**0.49**	**0.44**	0.01	0.06	**0.90**	0.22
Cluster Z4	**0.57**	0.24	0.03	**0.38**	0.17	0.03	0.10	0.04	0.05	**0.57**

Table 1. Example of a component partition matrix

4.4 Isolating the search cluster

After components have been assigned membership degrees to all the clusters and the final cluster centres have been determined, a user is able to search for components in the repository. The search procedure is not necessarily limited to one user at a time since each search only needs read-access to the cluster centres and membership degrees. Specifically, a user provides their preference through a simple user-interface application by selecting from the available categories in each attribute. The search preference is then transformed into a vector, just like in the form of the components in the repository, and is matched against the final cluster centres produced as a result of the previous clustering procedure. Subsequently, the cluster's index whose centre is nearest to the user's search preference is isolated to act as the "search cluster", based on the belief that it will inherently contain components nearer to the user's search preference. As always, the measure of closeness between a cluster centre and the search preference is calculated using the generalised Hamming distance. If a user decides not to give a category for certain attributes, then those attributes will not be taken into consideration during the matching with cluster centres and thus will be assessed based on a reduced attribute set. Furthermore, in the case where more than one cluster centre is nearest to the search preference (i.e., with equal distances from the search preference) then they are isolated and merged into one search cluster. Apart from their search preference, a

user can optionally select a value for a level of confidence, which represents a cut-off limit on the ranking of subsequently retrieved components.

4.5 Retrieval of software components

The final step of the methodology is to retrieve components from the repository and display the results to the user. The search results will include components contained only in the search cluster as this is considered to store software components nearest to the user's search preference. Furthermore, it is important that the retrieved results are ranked in order to allow the user to make a more informed reuse decision. The retrieval mechanism first calculates the membership degree of the search preference with respect to the search cluster. Therefore, components with similar degrees of participation in the search cluster will be near to the user's search preference. Thus, by isolating a range around the preference's degree of membership, the software components further away from the search preference can be eliminated whilst keeping those that are nearer. This range is constructed by defining an upper and lower bound based on the value of the preference membership degree. For experimentation purposes, this was set at ±10% meaning that, for instance, if a search preference was assigned a membership degree of 60% in the search cluster, the components retrieved would have membership degrees between 50%-70%. An example of this is presented in Fig. 3. In cases where the upper bound or lower bound is calculated to be higher than 100% or 0% respectively, the bounds are truncated accordingly.

The next stage of the retrieval procedure takes the subset of components and matches them against the user's search preference to find all the distances between the search preference and each component in the subset. This is done to form a ranking (in percent) from highest to lowest of the components based on the user's search preference matching attributes over the total number of attributes. However, before displaying the software components to the user, the level of confidence (given by the user in the second step of the methodology) is applied to eliminate those components that fall below a suitable level of similarity selected by the user. After the ranked set of components has been formed, the results of the search are ready to be displayed to the user.

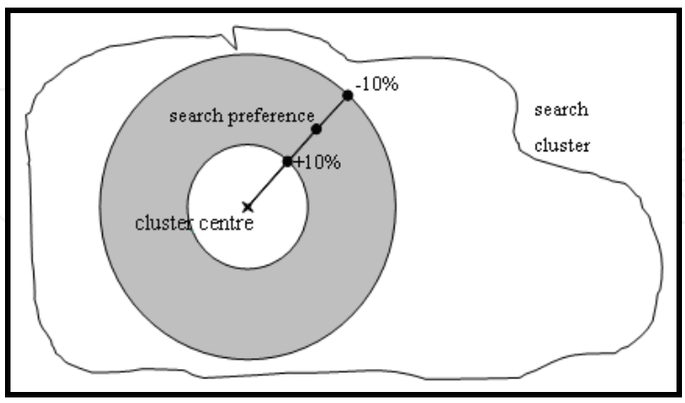


Fig. 3. Retrieval of the closest (suitable) components

The steps in the proposed methodology are outlined in Fig. 4. The methodology is shown as two separate units – one for the server (application administrator's) side and one for the client (user) side. Their only interaction is with the final cluster centres and the partition matrix. Of course, in reality, there would be more user instances connecting to the server.

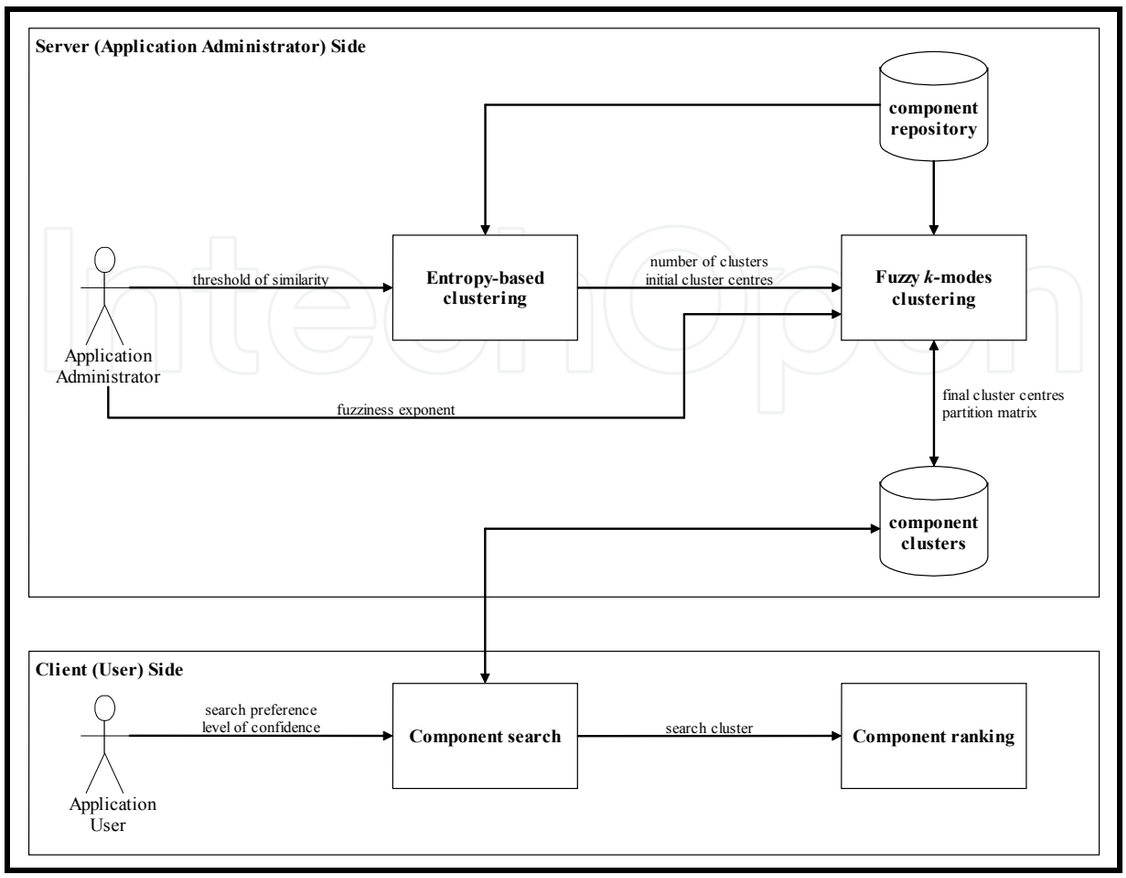


Fig. 4. Illustration of the proposed methodology

5. Experimental results

In order to test the efficiency and effectiveness of the methodology, a two-fold evaluation was carried, firstly to examine the clustering mechanism and secondly, to evaluate the retrieval mechanism.

5.1 Design of the experiments

The experiment involved the random generation of one-thousand software components and one random user search preference. For comparison reasons, the similarity of the search preference with all components was calculated and recorded, in order to distinguish which of these were closest to the search preference, as well as to be able to examine whether the clustering, isolation and retrieval processes do in fact return these closest components. The comparison was carried out according to the closest components in the dataset achieving similarity above: (1) 50% (near), (2) 75% (nearer) and (3) 90% (nearest). In addition, the methodology was tested by performing searches using three different variations of the preference: using all 15 attributes, 8 attributes and finally, 4 attributes.

For entropy-based clustering, an average threshold of similarity parameter (β = {0.50, 0.55}) was selected, bearing in mind the small number of attributes in the component dataset. Preliminary tests showed that with generally low total entropy values, a high threshold

leads to cases were, once the selection of the component with the lowest entropy value forms a cluster centre, very few or no components would achieve a higher similarity to the threshold in order to be assigned it. In this instance, clusters will only be composed of a single component – the centre itself – resulting further in the formation of an undesirably high number of cluster centres. As regards to fuzzy *k*-modes clustering, in particular, its fuzziness exponent, it was decided to use values of a = {1.10, 1.20} to control the level of fuzziness. Less strict values (i.e., high exponents) during the clustering procedure may lead to the overlooking of some of the suitable components during retrieval, which again is an undesirable outcome.

A prototype tool to support the methodology was built for both the application administrator (responsible for the clustering procedure) and the users/developers (for searching and retrieving). The tool, as with the implementation of the algorithms, was built using Matlab® Release 14, version 7.0.

5.2 Experimental results

A summary of the results are presented in Tables 2-4, displaying the number of components retrieved from the repository compared to the known number of components similar to the search preference.

The first evaluation (presented in Table 2) used all 15 attributes of the search preference to retrieve the most suitable components. With a "near" (>50%) similarity, it was calculated (using a simple pattern matching method) that 17 components were similar to the search preference. The best results were obtained with values β = 0.50 and a = 1.10, whereby the methodology managed to retrieve 13 of the 17 components, equalling a 76% match.

Similarity to search preference > 50% Number of known near components 17			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
13 (76%)	9 (53%)	11 (65%)	8 (47%)

Similarity to search preference > 75% Number of known nearer components 7			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
7 (100%)	7 (100%)	7 (100%)	7 (100%)

Similarity to search preference > 90% Number of known nearest components 2			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
2 (100%)	2 (100%)	2 (100%)	2 (100%)

Table 2. Percentage of the closest components retrieved using 15 attributes

Furthermore, it was observed that the 4 components not retrieved were actually not as close to the preference and that the most suitable components were indeed returned. In the

second case (similarity >75%) with 7 "nearer" components, all combinations managed to achieve a 100% match, retrieving all 7 components. Likewise, the 2 "nearest" components with respect to the search preference (similarity >90%) were both retrieved.

In the second evaluation only 8 of the 15 attributes of the search preference were used to retrieve components. In particular the attributes chosen were: domain functionality, specific functionality, operating system independence, implementation language, platform, visibility of implementation, price, and binding. The results are summarised in Table 3 below.

Similarity to search preference > 50% Number of known near components 45			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
12 (27%)	9 (20%)	10 (22%)	8 (18%)

Similarity to search preference > 75% Number of known nearer components 10			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
9 (90%)	8 (80%)	8 (80%)	8 (80%)

Similarity to search preference > 90% Number of known nearest components 2			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
2 (100%)	2 (100%)	2 (100%)	2 (100%)

Table 3. Percentage of the closest components retrieved using 8 attributes

By using only 8 attributes, it is observed that more components belong in the three similarity ranges (45 in >50%, 10 in >75% and 2 in >90%) than compared to using all 15 attributes. This is down to the fact that a fewer number of attributes decrease the likelihood of finding mismatches and hence greater distances. Moreover, from the results of Table 3, the percentage of "near" (> 50%) components with respect to the actual retrieved components has fallen substantially since retrieval is now based on a stricter matching. Also, although only 12 components were retrieved, these in fact were the ones that were closest to the preference, so in effect, the most suitable were not lost. At the two higher levels of similarity the best "nearer" and "nearest" components were almost always retrieved, with matches ranging from 80%-90% for similarity >75% and 100% for similarity >90% with the various combinations of parameters. Lastly, similarly to Table 2, the best results are obtained by using values of β = 0.50 and a = 1.10, which retrieve all 7 "nearest" components.

In the third and final evaluation, only the 4 attributes believed to be the most likely to be included in a search took part. These were the domain functionality, specific functionality, operating system independence, and implementation language. Table 4 shows the results obtained from the evaluation. It is immediately evident that the percentage of the "near" components retrieved is considerably low when compared to the other above experiments that used 15 or 8 attributes since the number of known components has increased. Specifically, even with the best β = 0.50 and a = 1.10 parameter values only 15 of the 87 (17%)

"near" components are retrieved and 9 of the 23 (39%) "nearer" components. Despite these low figures, in both cases, the components retrieved were still the ones closest to the preference. Also, all 7 of the "nearest" components were retrieved indicating at large that the methodology is capable of eliminating the less suitable components from the results at various levels of confidence.

Similarity to search preference > 50% Number of known near components 87			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
15 (17%)	9 (10%)	11 (13%)	8 (9%)

Similarity to search preference > 75% Number of known nearer components 23			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
9 (39%)	8 (35%)	8 (35%)	7 (30%)

Similarity to search preference > 90% Number of known nearest components 7			
β = 0.50		β = 0.55	
a = 1.10	a =1.20	a = 1.10	a =1.20
7 (100%)	6 (86%)	6 (86%)	6 (86%)

Table 4. Percentage of the closest components retrieved using 4 attributes

Based on the results of the various experiments it can be deduced that in order for the component retrieval process to be more accurate and reliable, the search must be carried out with as many, if not all the attributes defining the components. Nevertheless, this is attributable to the way in which the retrieval of the components works with respect to the quantity of the attributes – not quality, and certainly not due to the clustering procedure itself. Notably in the latter two experiments, although the number of components retrieved is low, they were still the most suitable according to their similarity to the search preference. As regards the retrieval of the "nearest" components, (similarity >90%), the methodology always manages to locate and display the best and most suitable components.

6. Conclusions

The number of software houses attempting to adopt a component-based development approach is rapidly increasing. However many organisations still find it difficult to complete the shift as it requires them to alter their entire software development process and philosophy. Furthermore, to promote component-based software engineering, organisations must be ready to promote reusability and this can only be attained if the proper framework exists from which a developer can access, search and retrieve a component. Hence, in the case of component-based software systems the ability to deliver software systems on time, within budget and with an acceptable level of quality is largely affected by the efficiency and effectiveness of the mechanisms employed for searching and retrieval of software

components. Component repositories, even though having a large impact on promoting component-based software development as an organised and accessible means for searching and retrieving software components, can be difficult and time-consuming to handle due to the usually large volume of residing components.

This chapter aspired to provide a methodology to improve the current means of searching and retrieving software components from a repository. Specifically, the proposed methodology employed computational intelligence techniques that seek to cluster software components based on their similarity and then continuing to search for components combining the suggested clustering scheme with an innovative mechanism for isolating and retrieving the closest and most suitable software components based on the search preference provided by its user.

6.1 Benefits and limitations

The proposed methodology has several advantages. Firstly, the clustering procedure employs highly simple, efficient and accurate algorithms, namely, the entropy-based and fuzzy *k*-modes clustering algorithms. Also, the methodology's scheme is very flexible since it can be easily adopted for use with any component repository, provided that the correct encoding of component attributes is applied. Furthermore, it can accommodate for an even higher-dimensional component representation (currently with 15 attributes) or even with a completely modified feature set.

On the other hand, the methodology is not without its drawbacks. One disadvantage lies in the fact that in reality, the results of the clustering algorithms rely on the subjectivity of the person performing the clustering, that is, the application administrator. Furthermore, it is their decision whether to keep a computed clustering scheme or to re-compute the final cluster centres. Also, there may be some cases where entropy-based clustering creates many clusters due to a low similarity of components in a repository. This would lead to a larger number for k and thus many final clusters. In this situation it is probable that a user's preference will be similar to many cluster centres and could lead to the search cluster containing too many clusters with which the preference will be compared too.

6.2 Discussion on future work

The number of parameters required in the methodology is relatively small. The application administrator is required to select values for the threshold of similarity of the entropy-based clustering algorithm and the fuzziness exponent of the fuzzy *k*-modes clustering algorithm. On the other hand, the user is only required to provide their search preference and level of confidence applied to the retrieved results. Even with its small number of input parameters in the clustering process, the values can have a significant impact on the search and retrieval of software components. One possible approach to eliminate this high dependence is to attempt to automatically calculate the most suitable values of these parameters instead of them having to be defined by the application administrator. This may be achieved either through pre-processing of the component repository or with a validation scheme after the clustering takes place. In (Tsekouras et al., 2005) the authors have proposed a cluster validity index that determines the optimal number of clusters based on compactness and separation criteria.

In general, the proposed methodology is functional and promising. Although, at first glance of the results, it seems to "lose" components during retrieval, in essence these components

are of lower similarity, and in fact, the methodology exhibits a strong filtering mechanism since it will always return the best (closest) suitable components from a repository for the user to evaluate and select which of these will be later integrated into the software system.

7. References

Agrawal, R.; Gehrke, J.; Gunopulos D. & Raghavan, P. (1998). Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 94-105, ISBN 0-89791-995-5, Seattle, WA, USA, June 1998, ACM Press, New York City

Andreou, A.S.; Vogiatzis, D.G. & Papadopoulos, G.A. (2006). Intelligent Classification and Retrieval of Software Components, *Proceedings of the Thirtieth Annual International Computer Software and Applications Conference, Vol. 2*, pp. 37-40, ISBN 0-7695-2655-1, Chicago, IL, USA, September 2006, IEEE Computer Society, Los Alamitos

Ankerst, M.; Breunig, M.M; Kriegel, H.-P. & Sander, J. (1999). OPTICS: Ordering Points to Identify the Clustering Structure, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 49-60, ISBN 1-58113-084-8, Philadelphia, PA, USA, June 1999, ACM Press, New York City

Bezdek, J.C. (1980). A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms, *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE)*, Vol. 2, No. 1, January 1980, 1-8, ISSN 0162-8828

Brown, A.W. & Wallnau, K.C. (1996). Engineering of Component-based Systems, *Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems*, pp. 414-422, ISBN 0-8186-7614-0, Montreal, Canada, October 1996, IEEE Computer Society, Los Alamitos

Chang, S.H.; Han, M.J. & Kim, S.D. (2005). A Tool to Automate Component Clustering and Identification, *Proceedings of the Eighth International Conference on Fundamental Approaches to Software Engineering*, pp. 141-144, ISBN 978-3-540-25420-1, Edinburgh, Scotland, April 2005, Springer-Verlag, Berlin

Chu, W.C.; Lu, C.-W.; Yang, H. & He, X. (2000). A Formal Approach for Component Retrieval and Integration Analysis, *Journal of Software Maintenance: Research & Practice*, Vol. 12, No. 6, November/December 2000, 325-342, ISSN 1532-060X

Cushner, K. & Brislin, R.W. (1997). *Improving Intercultural Interactions: Modules for Training Programs, Vol. 2*, ISBN 978-0761905370, Sage Publications, California

Ester, M.; Kriegel, H.-P.; Sander, J. & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226-231, ISBN 1-57735-004-9, Portland, OR, USA, August 1996, AAAI Press, Menlo Park

Frakes, W.B. (2007) Software Reuse, ReNews – Software Reuse and Domain Engineering, http://frakes.cs.vt.edu/renews.html

Girardi, M.R. & Ibrahim, B. (1995). Using English to Retrieve Software, *Journal of Systems & Software (Elsevier)*, Vol. 30, No. 3, September 1995, 249-270, ISSN 0164-1212

Capt. Haines, G.; Carney, D. & Foreman, J. (2007). Component-based Software Development/COTS Integration, *Software Technology Roadmap*, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, USA, http://www.sei.cmu.edu/str

Han, J. & Kamber, M. (2000) *Data Mining: Concepts and Techniques*, Morgan Kauffman, ISBN 1-55860-489-8, CA, USA

He, Z.; Xu, X. & Deng, S. (2007). Attribute Value Weighting in *k*-Modes Clustering, *Computer Science e-Prints: arXiv:cs/0701013v1 [cs.AI],* Cornell University Library, Cornell University, Ithaca, NY, USA, http://arxiv.org/abs/cs/0701013v1

Huang, Z. (1998) Extensions to the *k*-Means Algorithm for Clustering Large Datasets with Categorical Values, *Data Mining and Knowledge Discovery (Springer)*, Vol. 2, No. 3, September 1998, 283-304, ISSN 1384-5810

Huang, Z. & Ng, M.K. (1999). A Fuzzy *k*-Modes Algorithm for Clustering Categorical Data, *IEEE Transactions on Fuzzy Systems (IEEE)*, Vol. 7, No. 4, August 1999, 446-452, ISSN 1063-6706

Jain, A.K.; Murty, M.N. & Flynn, P.J. (1999). Data Clustering: A Review, *ACM Computing Surveys (ACM)*, Vol. 31, No. 3, September 1999, 264-323, ISSN 0360-0300

Kauffman, L. & Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley-InterScience, ISBN 978-0471878766, NY, USA

Kim, D.-W.; Lee, K.H. & Lee, D. (2004). Fuzzy Clustering of Categorical Data Using Fuzzy Centroids, *Pattern Recognition Letters (Elsevier)*, Vol. 25, No. 11, August 2004, 1263-1271, ISSN 0167-8655

Kohonen, T. (1987). *Content-Addressable Memories*, Springer-Verlag, ISBN 038717625X, NJ, USA

Naedele, M. (2000). Presentation Slides for Component Software: An Introduction, *Industrial Software Systems CHCRC.C2*, ABB Corporate Research Limited, Baden, Switzerland

Nakkrasae, S. & Sophatsathit, P. (2002). A Formal Approach for Specification and Classification of Software Components, *Proceedings of the Fourteenth International Conference on Software Engineering & Knowledge Engineering*, pp. 773-780, ISBN 1-58113-556-4, Ischia, Italy, July 2002, ACM Press, New York City

Nakkrasae, S.; Sophatsathit, P. & Edwards, Jr., W.R. (2004). Fuzzy Subtractive Clustering-based Index Approach for Software Components Classification, *International Journal of Computer & Information Science (ACIS)*, Vol. 5, No. 1, January 2004, 63-72, ISSN 1525-9293

Naur, P. & Randell, B. (1969). Software Engineering, *Report of a Conference Sponsored by the NATO Science Committee*, Garmisch, Germany, October 1968, Scientific Affairs Division, NATO, Brussels, Belgium

Prieto-Díaz, R. & Freeman, P. (1987). Classifying Software for Reuse, *IEEE Software (IEEE)*, Vol. 4, No. 1, January 1987, 6-16, ISSN 0740-7459

Prieto-Díaz, R. (1991). Implementing Faceted Classification for Software Reuse, *Communications of the ACM (ACM)*, Vol. 34, No. 5, May 1991, 89-97, ISSN 0001-0782

San, O.M.; Huyini, V.-N. & Nakamori, Y. (2004). An Alternative Extension of the *k*-Means Algorithm for Clustering Categorical Data, *International Journal of Applied Mathematics and Computer Science (AMCS)*, Vol. 14, No. 2, 2004, 241-247, ISSN 1641-876X

Sheikholeslami, G.; Chatterjee, S. & Zhang, A. (2000). WaveCluster: A Multi-resolution Clustering Approach for Very Large Spatial Databases, *International Journal on Very Large Data Bases (Springer)*, Vol. 8, No. 3-4, February 2000, 289-304, ISSN 1066-8888

Sugumaran, V. & Storey, V.C. (2003). A Semantic-based Approach to Component Retrieval, *The DATA BASE for Advances in Information Systems (ACM SIGMIS)*, Vol. 34, No. 3, August 2003, 8-24, ISSN 0095-0033

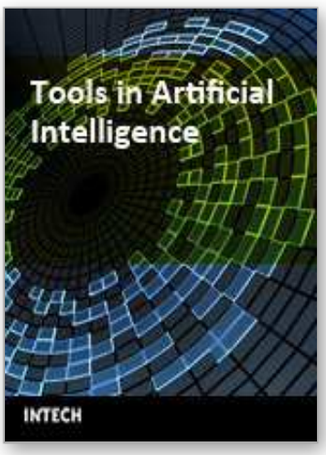
Tsekouras, G.E.; Papageorgiou, D.; Kotsiantis, S.; Kalloniatis, C. & Pintelas, P. (2005). Fuzzy Clustering of Categorical Attributes and its Use in Analyzing Cultural Data, *International Journal of Computing Intelligence (WASET)*, Vol. 1, No. 2, January 2005, 123-127, ISSN 1304-2386

Wang, W. ; Yang, J. & Muntz, R. (1997). STING: A Statistical Information Grid Approach to Spatial Data Mining, *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, pp. 186-195, ISBN 978-1558604704, Athens, Greece, August 1997, Morgan Kaufmann, San Francisco, California

Wang, Z.; Liu, D. & Feng, X. (2004). Improved SOM Clustering for Software Component Catalogue, *Proceedings of the International Symposium on Neural Networks, Vol. 1*, pp. 846-851, ISBN 978-3-540-22843-1, Dalian, China, August 2004, Springer-Verlag, Berlin

Yao, H. & Etzkorn, L. (2004). Towards a Semantic-based Approach for Software Reusable Component Classification and Retrieval", *Proceedings of the ACM Forty-second Annual Southeast Regional Conference*, pp. 110-115, ISBN 1-58113-870-9, Huntsville, AL, USA, April 2004, ACM Press, New York City

Yao, J.; Dash, M.; Tan, S.T. & Liu, H. (2000). Entropy-based Fuzzy Clustering and Fuzzy Modeling, *Fuzzy Sets and Systems (Elsevier)*, Vol. 113, No. 3, August 2000, 381-388, ISSN 0165-0114



Tools in Artificial Intelligence

Edited by Paula Fritzsche

ISBN 978-953-7619-03-9
Hard cover, 488 pages
Publisher InTech
Published online 01, August, 2008
Published in print edition August, 2008

This book offers in 27 chapters a collection of all the technical aspects of specifying, developing, and evaluating the theoretical underpinnings and applied mechanisms of AI tools. Topics covered include neural networks, fuzzy controls, decision trees, rule-based systems, data mining, genetic algorithm and agent systems, among many others. The goal of this book is to show some potential applications and give a partial picture of the current state-of-the-art of AI. Also, it is useful to inspire some future research ideas by identifying potential research directions. It is dedicated to students, researchers and practitioners in this area or in related fields.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen